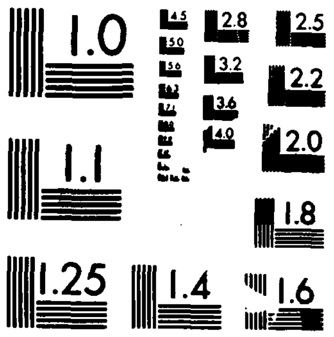
AD-A183 297   A GENERALIZED DBMS TO SUPPORT DIVERSIFIED DATA(U)   1/1
CALIFORNIA UNIV BERKELEY ELECTRONICS RESEARCH LAB
M STONEBRAKER 30 APR 85 AFOSR-TR-87-0944 AFOSR-83-0349

UNCLASSIFIED   F/G 12/7   NL


1.0
4.5
5.0
5.6
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

DTIC FILE COPY

2

AD-A183 297

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR. 87-0944	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Generalized DBMS to Support Diversified Data		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report (9/1/83 - 4/30/85)
		6. PERFORMING ORG. REPORT NUMBER
AUTHOR(s) Michael Stonebraker		8. CONTRACT OR GRANT NUMBER(s) AFOSR 83-0349
PERFORMING ORGANIZATION NAME AND ADDRESS Electronics Research Laboratory University of California Berkeley, CA 94720		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304 A3
CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research Bldg. 410, Bolling Air Force Base Washington, DC 20332 NM		12. REPORT DATE
		13. NUMBER OF PAGES 5
MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same as 11		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

unlimited

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

DTIC ELECTE AUG 1 9 1987 S D D

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

During the period of this grant we used the equipment bought under this proposal to support research on extendible type systems for data base use, main memory data bases and a new methodology for connecting application programs to data base systems. ~~We will~~ report on the research in these areas.

87

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AFOSR-TR- 87-0944

A GENERALIZED DBMS TO SUPPORT DIVERSIFIED DATA

Final Technical Report
AFOSR Grant 83-0349
(September 1, 1983 - April 30, 1985)

Michael Stonebraker
Principal Investigator

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California
Berkeley, CA 94720

A GENERALIZED DBMS TO SUPPORT DIVERSIFIED DATA

(final report)

Michael Stonebraker

EECS Department
University of California
Berkeley, Ca., 94720

1. INTRODUCTION

During the period of this grant we used the equipment bought under this proposal to support research on extendible type systems for data base use, main memory data bases and a new methodology for connecting application programs to data base systems. We will report on the research in these areas in the three sections which follow.

2. EXTENDIBLE TYPE SYSTEMS

Data bases must be designed to support the diverse data types found in non-business data processing applications. Such data elements include test, bit maps, polygons, spatial objects and time. As noted in [STON83] such objects are very inefficient to support on top of the built-in data types supplied by existing data base systems. What is needed is a mechanism to add user defined data types to a data base system. What follows is a brief sketch of the solution we have explored. A full treatment of this topic appears in [STON86].

A user can create a new data type by indicating to the DBMS:

The name of the type
A procedure to convert an ASCII data element to one of the indicated type
A procedure to convert an element from the indicated type to ASCII
The storage length of the type

With this information, data elements of the new type can be stored in the data base and retrieved. To achieve real leverage, operators and functions should also be definable on the new type. We will discuss operators first. Query languages support unary and binary operators, and current query optimizers can generate optimized plans for query language expressions containing such operators. The main problem to be solved is that of constructing a query optimizer that can successfully process query language expressions with user defined operators.

The solution is to convert a query optimizer to be table driven and to require each operation that is defined to specify the required information. We will indicate the approach with a brief example. Most data base systems include merge-sort as an

A-1

optimization tactic for queries that include joins. Using merge-sort, both relations are sorted on the field used in the join expression and then the sorted relations are merged to produce the answer. A current query optimizer has built into it the knowledge that "=" is a join operator for which merge-sort is usable and that both relations must be sorted using the "<" operator. An optimizer that processes extendible operators must be able to know for each operator whether merge-sort is a usable tactic and what sort operator to use in the algorithm. This information can be specified when each operator is defined. Then, the run-time system must be able to process merge-sort by calling a sort package which can sort using any operator. The other changes that a DBMS optimizer must undergo are specified in [STON86], and an implementation of these ideas is underway [STON86A].

The second thing which an extendible type system requires is the possibility of user defined indexes. Current DBMSs give users B-trees and sometimes hashing as access methods. Such methods are ideal for business data processing applications, but fail completely when presented with spatial data. A DBMS must thereby be capable of adding user definable access methods. Candidates for spatial data include R-trees [GUTM84] and K-D-B-trees [ROBI81].

The main problem in supporting new access methods is instructing the query optimizer on the performance of such new paths to data. One must be able to tell the optimizer what form of expressions the access methods can process and what the expected cost of any such expression is. The solution is to include the expressions in a compact notation when any given access path is created. The expected cost of any expression is indicated by entries in appropriate system catalogs. The query optimizer can then look in the catalogs for needed information. Complete details of this technique are given in [STON86].

3. MAIN MEMORY DATA BASES

In engineering applications one frequently browses a large amount of related data. Hence, the plummeting cost of semiconductor memory makes it increasingly possible to put a data base entirely in primary memory. As part of this research project we investigated techniques to apply main memory data.

First, we investigated how well B-trees performed when compared with structures exclusively designed for main memory use such as balanced binary trees. When data fits entirely in main storage, binary trees were shown to be superior; however, their performance degraded rapidly when insufficient main storage was available for the size of the data set. We concluded that main memory oriented structures were only suitable if virtually the entire data set fits in main memory. The details of this analysis were presented in [DEWI84].

The second aspect of main memory which we investigated was join tactics that could be used with large amounts of main memory. We analyzed several algorithms, including conventional range sort, a basic join algorithm based on hashing, and two algorithms based on partitioned hashing. Our analysis showed that when a small amount of main memory was available, the merge-sort was the clear winner. When storage was available for the smaller of the two relations, then any of the hash join algorithms outperformed merge join by a substantial margin. Between those two amounts of memory, partitioned hashing was shown to be advantageous. Again the details of the analysis appear in [DEWI84].

The final aspect of a main memory data base system which we studied was the processing of data base logs. In a main memory environment, the I/Os associated with the log may be one of the most expensive aspects of transaction processing. First, we investigated the possibility of processing a whole collection of transaction commit operations as a group. This "group commit" concept was shown to dramatically enhance performance and was independently investigated by the designers of the IMS/Fast Path System. We also studied the speedups available by utilizing persistent main memory, i.e., memory that does not fail when the power is interrupted or a crash occurs. It was shown that moving the log to persistent main memory was an advantageous tactic.

4. INTERFACE TO AN APPLICATION PROGRAM

Conventional data base systems use the notion of a "cursor" to interface between a data manager and an application program. An application can define a query and then iterate over the records that it produces using this construct. However, especially in browsing applications the cursor concept has at least three disadvantages. First, cursors only allow an application to retrieve records in a forward direction. After a record has been fetched, there is no easy way to move backwards and fetch it again as might be done by a typical browsing application. The second problem is that only one record can be fetched at a time. Here an application that fills up a screen with 20 or 30 records must fetch them one by one. The last disadvantage is that once a query is specified, there is no way to dynamically restrict its scope. For example, while browsing the employees whose age is less than 40, one cannot search forward to an employee named Smith except by scanning all records one by one and searching for the qualifying name. This makes browsing-oriented programs difficult to write and subject to bad performance.

We have designed a new interface based on the concept of a portal. A portal functions much like a "tall cursor," i.e., one for a which a variable number of records are returned on each call. Portals allow forward and backward scans as well as forward and backward search operations. As such they are a much better base on which to build browsing oriented application programs. A completed proposal of the portal concept appears in [STON84] and we are implementing a variant of this proposal in [STON86A].

REFERENCES

[DEWI84]	Dewitt, D. et. al., "Implementation Techniques for Main Memory Data Bases, "Proc. 1984 ACM-SIGMOD Conference on Management of Data, Boston, Mass., June 1984.
[GUTM84]	Gutman, A., "R-trees: A Dynamic Index Structure for Spatial Searching," Proc. 1984 ACM-SIGMOD Conference on Management of Data, Boston, Mass. June 1984.
[ROBI81]	Robinson, J., "The K-D-B Tree: A Search Structure for Large Multidimensional Indexes," Proc. 1981 ACM-SIGMOD Conference on Management of Data, Ann Arbor, Mich., May 1981.
[STON83]	Stonebraker, M., et al., "Application of Abstract Data Types and Abstract Indexes to CAD Data," Proc. Engineering Applications Stream of 1983 Data Base Week, San Jose, Ca., May 1983.

[STON84] Stonebraker, M. and Rowe, L.A., "PORTALS: A New Application Program Interface," Proc. 1984 VLDB Conference, Singapore, Sept. 1984.

[STON86] Stonebraker, M., "Inclusion of New Types in Relational Data Base Systems," Proc. Second International Conference on Data Base Engineering, Los Angeles, Ca., Feb. 1986.

[STON86A] Stonebraker, M. and Rowe, L., "The Design of POSTGRES," Proc. 1986 ACM-SIGMOD Conference on Management of Data, Washington, D.C., May 1986.

END
9-87
DTIC